

Automating the Analysis of Planetary Nebulae Images

Bernd Fischer

Johann Schumann

RIACS / NASA Ames Research Center

E-mail: {fisch,schumann}@email.arc.nasa.gov

Abstract

Scientific data analysis involves the task of fitting statistical models of the studied processes to collected data. Here, we take a typical task, the analysis of planetary nebulae images taken by the Hubble Space Telescope, and describe how the program synthesis system AUTOBAYES can be used to generate the necessary analysis programs. We describe the AUTOBAYES system and discuss its fully declarative model specification language. We present the automatic derivation of the scientists' original analysis [12] as well as a refined analysis using image segmentation models. This demonstrates that the AUTOBAYES synthesis system can be applied to realistic scientific data analysis tasks.

1 Introduction

Planetary nebulae are remnants of dying stars. Scientists try to understand their physics by collecting and analyzing data, for example images taken by the Hubble Space Telescope (HST). The analysis follows a general pattern in science: the scientists formulate their initial understanding of the underlying physical processes as a model, fit the model to the collected data (i.e., estimate the values of the model parameters of interest from the data), interpret the results, and refine the model as long as necessary. Since both the underlying processes and the data collection are fraught with uncertainty and noise, statistical models are used.

In most disciplines, the large data volumes collected by modern instruments make computer support indispensable. Consequently, development and refinement of the necessary data analysis programs have become a bottleneck, and it is not unheard that tapes of unanalyzed data sit in warehouses for several years before the software is finally completed. To increase the speed with which reliable data analysis software can be developed, we are currently building the AUTOBAYES system.

AUTOBAYES is a fully automatic program synthesis system for data analysis problems. Its input is a declarative problem description in form of a statistical model; its out-

put is documented and optimized C/C++ code. Its schema-based approach allows the use of advanced algorithms and data structures and yields fast turnaround times comparable to compilation times, supporting the iterative development style typical for the domain. AUTOBAYES thus enables the scientists to think and to program in models instead of code.

In this paper, we take one typical scientific data analysis problem—the analysis of planetary nebulae images taken by the HST—and show that and how AUTOBAYES can be used to automate the implementation of the necessary analysis programs. We initially follow the analysis described in [12] and use AUTOBAYES to derive code for the models published by the space scientists. We then go beyond their analysis and use AUTOBAYES to derive code for a simple image segmentation procedure which can be used to automate a manual preprocessing step done by the scientists. Finally, we combine the original approach with the simple image segmentation which yields a much more detailed analysis. This also demonstrates that AUTOBAYES makes it easy to combine different aspects of data analysis.

The main contribution of this paper is to demonstrate that program synthesis in general and AUTOBAYES in particular have reached a level of maturity which make them applicable to realistic scientific data analysis applications. Moreover, we show that synthesis enables scientists to formulate a more detailed analysis.

2 Background

2.1 Planetary Nebulae

Stars with initial masses between roughly 0.8 and 8 solar masses turn into swollen red giants when they run out of hydrogen to support their primary fusion process. In a secondary fusion process, these giants then burn the helium produced by the hydrogen fusion, resulting in a carbon-oxygen core roughly the size of the earth. Eventually, the secondary fusion runs out of fuel as well and the red giants begin to collapse into extremely hot white dwarfs. During this collapse, most of the material is expelled, forming blown-out gaseous shells which are called planetary neb-

ulae. The shells continue to expand and after 10,000 to 50,000 years their density becomes too small for the nebulae to be visible.



Figure 1. False-color image of IC418

Figure 1 shows a composite false-color image of the planetary nebula IC418, also known as the Spirograph Nebula, taken by the HST (Sahai et al., NASA and The Hubble Heritage Team). The different colors (resp. gray-scales) indicate the different chemicals prevalent in the different regions of the nebula; the origin of the visible texture is still unknown. The central white dwarf is discernible as a white dot in the center of the nebula.

Planetary nebulae occupy an important position in the stellar lifecycle and are the major sources of interstellar carbon and oxygen but their physics and dynamics are not yet well understood. The characterization and analysis of their properties is thus an important task in astronomy.

2.2 AUTOBAYES

AUTOBAYES[10, 11] is a fully automatic program synthesis system for data analysis problems.¹ Externally, it looks like a compiler: it takes an abstract problem specification and translates it into executable code. Internally, however, it is quite different: AUTOBAYES first derives a customized algorithm implementing the model and then optimized, imperative code implementing the algorithm. AUTOBAYES is implemented in SWI-Prolog² and currently comprises about 64,000 lines of documented code. Figure 2 shows the system architecture; in the following we explain the major components.

Statistical Models and Specification Language. A *statistical model* describes the expected properties of the data in a fully declarative fashion: for each problem variable of interest (i.e., observation or parameter), properties and dependencies are specified via probability distributions and constraints. Figure 3 shows how a model (discussed in more detail in Section 3.1) is represented in AUTOBAYES's specification language.³ Line 1 just identifies the model. Lines

¹AUTOBAYES is not yet available publicly; for a web-based interface and other papers see <http://ase.arc.nasa.gov/autobayes>.

²<http://www.swi-prolog.org>

³Keywords have been underlined and line numbers have been added for reference; comments start with a % and extend to the end of the line.

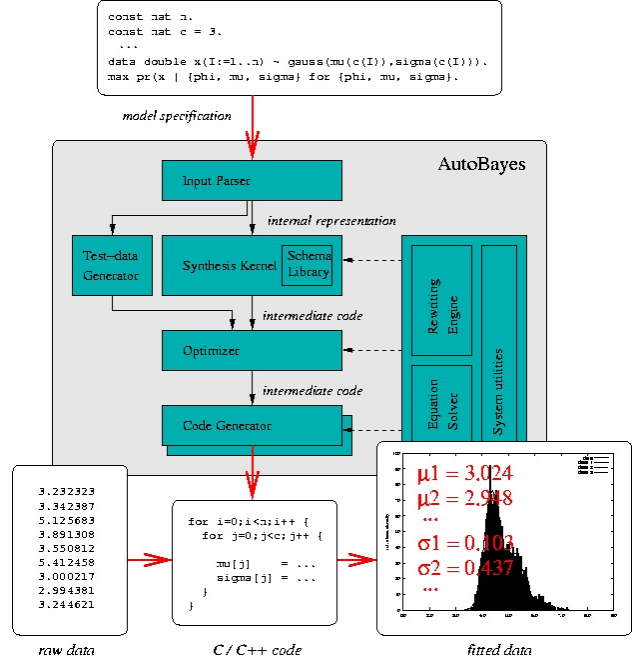


Figure 2. AUTOBAYES system architecture.

2 and 4 introduce symbolic constants whose values are left unspecified but constrained by the where-clauses in lines 3 and 5, respectively. In general, constraints can be complex boolean formulae tying together multiple variables (cf. line 11). Lines 6–14 introduce the parameters, again constrained by where-clauses. Variables can be annotated with *as*-clauses; these textual annotations are propagated into the generated code to improve its legibility. Line 16 declares the observation (denoted by the data-modifier) as a matrix; its expected properties are specified in the distribution clause in line 17. In general, a distribution clause is of the form $x \sim D(\vec{y})$ where x is a single variable or vector/matrix element, D is a distribution, and \vec{y} are the distribution's parameters. Distributions can be both discrete (e.g., binomial) or continuous (e.g., Gaussian, Poisson, ...) but have to be univariate; multivariate distributions (i.e., functions in $\mathbb{R}^N \rightarrow \mathbb{R}^M$) are not yet supported by AUTOBAYES. Distributions are chosen from a predefined list; adding more distributions is straightforward. The final line in the model is the *task* clause. It specifies the analysis problem the synthesized program has to solve. Since AUTOBAYES only supports *parameter learning* (i.e., the estimation of the parameter values best explaining the observed data, given a model) but not *structure learning* (i.e., the estimation of the best model itself), tasks have the form to maximize a conditional probability w.r.t. a set of goal variables. In this case, the task is a *maximum likelihood estimation* because all goal variables occur to the right of the conditioning bar in the conditional probability and there are no priors on the parameters.

```

1 model gauss as '2D Gauss-Model for Nebulae Analysis'.
% Image size
2 const nat nx as 'number of pixels, x-dimension'.
3   where 0 < nx.
4 const nat ny as 'number of pixels, y-dimension'.
5   where 0 < ny.
% Center; assume center is on the image
6 double x0 as 'center position, x-dimension'.
7   where 1 =< x0 && x0 =< nx.
8 double y0 as 'center position, y-dimension'.
9   where 1 =< y0 && y0 =< ny.
% Extent; assume full nebula is on the image
10 double r as 'radius of the nebula'.
11   where 0 < r && r < nx/2 && r < ny/2.
% Intensity; upper bound determined by instrument
12 double i0 as 'overall intensity of the nebula'.
13   where 0 < i0 && i0 =< 255.
% Noise; upper bound arbitrary, for initialization
14 double sigma as 'noise'.
15   where 0 < sigma && sigma < 100000.
% Data and Distribution
16 data double pixel(1..nx, 1..ny) as 'image'.
17 pixel(I,J) ~ gauss(i0 * exp(-((I-x0)**2 + (J-y0)**2)
                     / (2*r**2)),
                     sigma).
% Task
18 max pr(pixel|{i0,x0,y0,r,sigma}) for {i0,x0,y0,r,sigma}.

```

Figure 3. Complete AUTOBAYES-specification for Gaussian model.

The application domain only uses restricted datatypes; hence, the specification language currently supports only the three basic types `nat`, `int`, and `double`, with array as the single type constructor. However, a more expressive type system could be added.

Bayesian Networks. A *Bayesian network* is a directed, acyclic graph whose nodes represent random variables and whose edges define probabilistic dependencies between the random variables. AUTOBAYES uses them to represent the statistical models internally. Figure 4 shows an example network automatically extracted from the Gaussian model specification and drawn using the `dot` graph layoutter.

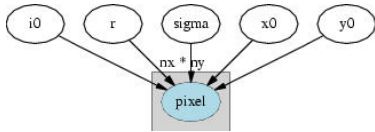


Figure 4. Bayesian net for Gaussian model.

AUTOBAYES uses a variant of *hybrid* Bayesian networks, where nodes can represent discrete as well as continuous random variables; these are rendered as boxes and ellipses, respectively. However, here all variables are continuous. Shaded nodes represent known variables, i.e., input data. Shaded boxes enclosing a set of nodes represent *plates* [5], i.e., collections of independent, co-indexed random variables. Distribution information for the random variables is attached to the respective nodes. Here, *pixel* is a $nx \times ny$ matrix of independent and identically distributed

(i.i.d.) Gaussian random variables with observed values.

Bayesian networks combine probability theory and graph theory. They are a common representation method in machine learning [5, 18] because they provide an efficient encoding of the joint probability distribution over all variables and thus allow to replace expensive probabilistic reasoning by faster graphical reasoning.

Schemas and Schema Library. Program synthesis from first principles (i.e., purely deductive synthesis) is notoriously difficult to scale up (cf. [3, 13]). AUTOBAYES thus follows a schema-based approach. A *schema* consists of a parameterized code fragment (i.e., template) and a set of constraints. The parameters are instantiated by AUTOBAYES, either directly or by calling itself recursively with a modified problem. The constraints determine whether a schema is applicable and how the parameters can be instantiated. Constraints are formulated as conditions on the Bayesian network or directly on the specified model; they include the task clause as special case. This allows the network structure to guide the application of the schemas and thus to constrain combinatorial explosion of the search space, even if a large number of schemas is available. Schemas can in principle be understood as conditional rewrite rules on partially instantiated programs, where the only redexes are maximization tasks. They are implemented as Prolog-clauses and search control is thus simply relegated to the Prolog-interpreter: schemas are tried in their textual order. This simple approach has not caused problems so far, mainly because the domain admits a natural layering which can be used to organize the schema library. The top layer comprises network decomposition schemas which try to break down the network into independent subnets, based on independence theorems for Bayesian networks. These are domain-specific divide-and-conquer schemas: the emerging subnets are fed back into the synthesis process and the resulting programs are composed to achieve a program for the original problem. AUTOBAYES is thus able to automatically synthesize larger programs by composition of different schemas. The next layer comprises more localized decomposition schemas which work on products of i.i.d. variables. Their application is also guided by the network structure but they require more substantial symbolic computations. The core layer of the library contains statistical algorithm schemas as for example *expectation maximization* (EM) [7, 14] and k-Means (i.e., nearest neighbor clustering); these generate the skeleton of the program. The final layer contains standard numeric optimization methods as for example the Nelder-Mead simplex method or different conjugate gradient methods. These are applied after the statistical problem has been transformed into an ordinary numeric optimization problem and AUTOBAYES failed to find a symbolic solution for the problem. Currently, the library comprises 28 top-level schemas, with a number of

additional variations (e.g., different initializations).

Symbolic Subsystem. AUTOBAYES relies significantly on symbolic computations to support schema instantiation and code optimization. The core part of the symbolic subsystem implements symbolic-algebraic computations, similar to those in Mathematica. It is based on a small but reasonably efficient rewrite engine which supports associative-commutative operators and explicit contexts. Hence, AUTOBAYES allows contextual rules (i.e., conditional rules accessing an explicit context) as for example $x/x \rightarrow_{C \vdash x \neq 0} 1$ where $\rightarrow_{C \vdash x \neq 0}$ means “rewrites to, provided $x \neq 0$ can be proven from the current context C .” The contexts are managed almost transparently by the rewrite engine; rewrite systems only need to contain the non-congruent propagation rules which modify the contexts under which immediate subterms are rewritten, e.g., $p ? s : t \xrightarrow{\sim}_C (p \downarrow_C) ? (s \downarrow_{C \wedge p}) : (t \downarrow_{C \wedge \neg p})$ for C-style conditionals. Here, $\xrightarrow{\sim}_C$ and \downarrow_C denote context propagation from and normal form computation under the context C , respectively.

Expression simplification and symbolic differentiation are implemented on top of the rewrite engine. The basic rules are straightforward; however, vectors and matrices introduce the usual aliasing problems and require careful formalizations. For example, as the index values i and j are usually unknown at synthesis time, the partial derivative $\partial x_i / \partial x_j$ can only be rewritten into $i = j ? 1 : 0$. Some rules even require explicit meta-programming, in particular when bound variables are involved. In total, AUTOBAYES’s symbolic system contains 365 rewrite rules.

Abstract interpretation is used as to efficiently evaluate range constraints such as $x > 0$ or $x \neq 0$ which occur in the conditions of many rewrite rules. AUTOBAYES implements as a rewrite system a domain-specific refinement of the standard sign abstraction where numbers are not only abstracted into *pos* and *neg* but also into *small* (i.e., $|x| < 1$) and *large*.

It then turns out that a relatively simple solver built on top of this core system is already sufficient. AUTOBAYES thus essentially relies on a low-order polynomial (i.e., linear, quadratic, and simple cubic) symbolic solver. However, it also shifts and normalizes exponents, recognizes multiple roots and bi-quadratic forms, and tries to find polynomial factors. It also handles expressions in x and $(1 - x)$ which are common in statistical applications.

A smaller part of the symbolic subsystem implements the graphical reasoning routines necessary for Bayesian networks with plates, for example, computing the parents, children, or Markov blanket [18] of a node.

Intermediate Code. The code fragments in AUTOBAYES’ schemas are written in an imperative intermediate language. This is essentially a “sanitized” variant of C (i.e., no pointers, side effects in expressions etc.), similar in spirit

to CIL [16]. Unlike CIL, however, the AUTOBAYES intermediate code also contains a number of domain-specific constructs like vector/matrix operations, finite sums, and convergence-loops.

Optimization. Straightforward schema instantiation and composition produces suboptimal code; worse, many of the suboptimalities cannot be removed completely using a separate, after-the-fact optimization phase. AUTOBAYES thus interleaves synthesis and optimization. Schemas can explicitly trigger large-scale optimizations which take into account information from the synthesis process. For example, all numeric optimization routines restructure the goal expression using code motion, common sub-expression elimination, and memoization; since the schemas know the goal variables, no dataflow analysis is required to identify invariant sub-expressions, and code can be moved around much more aggressively, even across procedure borders. Schemas can also generate dynamic rewrite rules (similar to dynamic rules in Stratego [22]) which are applied whenever symbolic simplification is used.

A final rewrite-based optimization pass re-introduces eliminated operators like subtraction and division (“denormalization”) to improve expression evaluation and performs a number of peephole optimizations to improve the legibility of the generated code. Currently, “traditional” dataflow-oriented optimizations like constant propagation or loop fusion are left to the compiler; however, such optimizations can also be encoded in a rewrite-based approach [17].

Code Generation. In a final step, AUTOBAYES translates the optimized intermediate code into code tailored for a specific run-time environment. Currently, AUTOBAYES has code generators for the Octave and Matlab environments; it can also produce standalone C and Modula-2 code.

Each code generator employs one rewrite system to eliminate the constructs of the intermediate language which are not supported by the target environment (“desugaring”) and a second rewrite system to clean up the desugared code; most rules are shared between the different code generators. The actual surface syntax is then produced by a straightforward pretty-print of the syntax tree.

Certification. Schema-based synthesis approaches cannot ensure “correctness-by-construction” the same way as purely deductive approaches. Since formally verifying the entire system is unfeasible, we certify each generated program individually. We concentrate on specific aspects of program safety (e.g., memory safety) which are formalized in a safety policy as a set of Hoare-rules [23, 8]. The schemas are extended to generated code and all required annotations such that a verification condition generator can produce proof obligations; these are then discharged using an automated theorem prover.

Test-Data Generation. Statistical models also allow to generate problem-specific test data (i.e., *sampling*), if the

the edges of the extracted network are followed forward from the sources and random numbers are generated along the way. AUTOBAYES uses this interpretation to generate model-specific sampling programs which can be used to validate the models and solutions.

3 A Hierarchical Set of Models

Knuth and Hajian [12] have presented a hierarchical set of models they used to analyze the images of planetary nebulae. Each of the three models estimates a parameter set which is then refined by the subsequent models. Here we show how these models are represented in AUTOBAYES' specification language, and how the code is derived.

3.1 Gaussian Model

The initial analysis task is to identify the position of the center of nebula on the image; this also serves as starting point for all subsequent analysis tasks. Many quick-and-dirty solutions (e.g., computing the center of the light mass over the entire image) can be devised for this task, but using AUTOBAYES even a statistically clean solution can be obtained quickly.

The two core ideas of all the models presented by Knuth and Hajian are (i) that the light intensity which is to be expected at a given pixel position (x, y) on the image can be described by a function F of this position, the (unknown) center (x_0, y_0) of the nebula and, depending on the function, some additional parameters, and (ii) that the measured intensities can be fitted against this function using a simple mean square error minimization. The only difference between the three models is the form of F . In the first model, F has the shape of a bell whose apex is at (x_0, y_0) which can be formalized by a two-dimensional Gaussian curve:

$$F(x, y) = i_0 \cdot e^{-\frac{(x_0-x)^2+(y_0-y)^2}{2r^2}} \quad (1)$$

The additional parameters i_0 and r capture the overall intensity and extent of the nebula (i.e., the height and diameter of the bell).

Model Specification. The AUTOBAYES specification shown in Figure 3 is a direct transcription of the underlying mathematics. The distribution clause for the image pixels in line 17 formalizes the idea that the expected value of the pixel (i, j) can be described by the function $F(x, y)$ from Equation (1); remember that the expected value of a Gaussian random variable is given by the mean (i.e., first parameter) of the distribution. The standard deviation (i.e., second parameter) of the distribution represents the error of the fit. In combination with the Gaussian distribution, the task clause in line 18 thus specifies a mean square error minimization. The constraints formalize additional as-

sumptions on the structure of the image or the output of the instrument (cf. line 13).

Mathematical Derivation. For the Gaussian model, the program derivation can neatly be separated into two phases such that the first phase is a purely mathematical derivation and the second phase only instantiates code templates; in general, however, this is not the case and symbolic computation and template instantiation are interleaved.

The first step is to unfold the entire *pixel*-matrix element by element, using a decomposition schema based on the conditionalized version of the general product rule for probabilities:

$$\begin{aligned} pr(\mathbf{pixel} \mid i_0, x_0, y_0, r, \sigma) \\ = \prod_{i=1}^{nx} \prod_{j=1}^{ny} pr(pixel(i, j) \mid i_0, x_0, y_0, r, \sigma) \end{aligned}$$

The precondition for this step is that the pixels are pairwise independent, given the remaining variables, i.e., that

$$\begin{aligned} pr(pixel(i, j) \mid pixel(i', j'), i_0, x_0, y_0, r, \sigma) \\ = pr(pixel(i, j) \mid i_0, x_0, y_0, r, \sigma) \end{aligned}$$

holds for all i, j, i', j' such that $i \neq i'$ or $j \neq j'$. Instead of proving this from first principles, using the distribution information given in the model specification, AUTOBAYES can easily check it on the Bayesian network: there is no edge going from the *pixel*-node into itself and, hence, by definition of Bayesian networks, the pixels are pairwise independent.

The probability is now elementary in the sense that on the left of the conditioning bar we have only a single variable $pixel(i, j)$ which depends exactly on all the variables to the right of the conditioning bar. Hence, the probability expression can be replaced by the distribution function. AUTOBAYES's domain theory contains rewrite rules for the most common probability density functions; additional density functions can easily be added. This rewrite yields the likelihood-function

$$\prod_{i=1}^{nx} \prod_{j=1}^{ny} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{\left[pixel(i, j) - i_0 \cdot e^{-\frac{(x_0-i)^2+(y_0-j)^2}{2r^2}} \right]^2}{2\sigma^2}}$$

which must be maximized w.r.t. goal variables i_0, x_0, y_0, r , and σ . In general it is easier to work with the log-likelihood function which yields the same solutions during maximization since the logarithm is strictly monotone. After simplification using the symbolic subsystem, AUTOBAYES thus derives the following log-likelihood function:

$$\begin{aligned} L = & -nx \cdot ny \cdot \log(2\pi) - nx \cdot ny \cdot \log(\sigma) - \\ & \frac{1}{2\sigma^2} \cdot \sum_{i=1}^{nx} \sum_{j=1}^{ny} \left[pixel(i, j) - i_0 \cdot e^{-\frac{(i-x_0)^2+(j-y_0)^2}{2r^2}} \right]^2 \end{aligned}$$

A solution can now be attempted in two different ways, numerically or symbolically. By default, AUTOBAYES tries to

find symbolic solutions first. In this case, it computes the partial differentials

$$\begin{aligned}\frac{\partial L}{\partial i_0} &= \frac{1}{\sigma^2} \cdot \sum_{i=1}^{nx} \sum_{j=1}^{ny} \text{pixel}(i, j) \cdot e^{-\frac{(i-x_0)^2 + (j-y_0)^2}{2r^2}} \\ &\quad - \frac{i_0}{\sigma^2} \cdot \sum_{i=1}^{nx} \sum_{j=1}^{ny} e^{-\frac{(i-x_0)^2 + (j-y_0)^2}{r^2}} \\ \frac{\partial L}{\partial \sigma} &= \frac{1}{\sigma^3} \cdot \sum_{i=1}^{nx} \sum_{j=1}^{ny} \left[\text{pixel}(i, j) - i_0 \cdot e^{-\frac{(i-x_0)^2 + (j-y_0)^2}{2r^2}} \right]^2 \\ &\quad - \frac{nx \cdot ny}{\sigma}\end{aligned}$$

and solves the equations

$$\frac{\partial L}{\partial i_0} \stackrel{!}{=} 0 \quad \frac{\partial L}{\partial \sigma} \stackrel{!}{=} 0$$

which are essentially simple polynomials in i_0 and σ . These can easily be handled by the built-in equation solver; however, attempts to solve for the remaining three variables x_0 , y_0 , and r fail.

Code Derivation. At this point, the symbolic computations have been exhausted without leading to a complete symbolic (i.e., closed-form) solution. AUTOBAYES thus derives code for a numeric solution, incorporating the computed partial symbolic solution. This is done in two steps, which correspond to schemas in the schema library.

In the first step, AUTOBAYES converts the symbolic solutions into assignment statements. Then it identifies their order and position relative to the remaining code which is still to be synthesized. Since both solutions contain at least one (in fact all) of the remaining variables, they must follow that code: since the solution for σ contains i_0 , its assignment must in turn follow that of i_0 . AUTOBAYES then eliminates them from the formula by applying the substitution corresponding to the solution. Variables whose solutions do not depend on any unsolved variable can be considered as symbolic constants and need not be eliminated; their corresponding assignments must precede the missing code block. Since this reasoning is done on the (side-effect free) expression level, a dataflow analysis is not required.

In the second step, AUTOBAYES instantiates a numeric optimization routine, in this case the Fletcher-Reeves conjugate gradient method. The schema actually contains only a wrapper to the implementation provided by the GNU Scientific Library (GSL).⁴ However, this wrapper is not just boilerplate code because it contains specific initialization code and a number of auxiliary functions to evaluate the goal function and the derivatives. AUTOBAYES contains different heuristics to derive initialization code from specification information; here, the initial values are taken as the midpoints of the specified ranges. AUTOBAYES also generates the auxiliary functions; since a straightforward translation from the goal expression would be prohibitively inefficient, AUTOBAYES aggressively optimizes the functions. The optimizations include common subexpression

⁴<http://sources.redhat.com/gsl/>

elimination, memoization, and code motion, and are applied both intra- and inter-procedural (although the latter restricted to the generated auxiliary functions). The optimizations can also take into account locally constant variables, since AUTOBAYES knows the set of goal variables. Again, a dataflow analysis is not required since the reasoning is done on the expression-level.

Program Results. We have applied the generated program to the manually masked image of IC418 shown in the second panel of Figure 5. The third panel shows the results. The program roughly approximates the center but its estimate of the overall extent is predictably off the mark.

3.2 Sigmoidal Models

With the choice of the function $F(x, y)$, the Gaussian model presented above hard-codes a number of assumptions about planetary nebulae or, more precisely, about the structure of the images; in particular, it assumes that the images are circular, with a pronounced intensity peak and a gradual intensity falloff at the edges. However, a quick look at Figure 5 shows that this is only a coarse approximation: the image of IC418 is clearly more elliptic than circular, and the intensity shows rather a broad plateau with a pronounced falloff at the edges than the other way round.

Simple Sigmoidal Model. Knuth and Hajian thus refine their initial model and replace the two-dimensional Gaussian by a two-dimensional sigmoidal function of the form

$$F(x, y) = i_0 \cdot \left[1 - \frac{1}{1 + e^{-a\sqrt{r(x, y)-1}}} \right] \quad (2)$$

with the auxiliary function $r(x, y)$

$$r(x, y) = c_{xx} \cdot (x_0 - x)^2 + 2c_{xy} \cdot (x_0 - x)(y_0 - y) + c_{yy} \cdot (y_0 - y)^2$$

and constants c_{xx} , c_{yy} , and c_{xy}

$$\begin{aligned}c_{xx} &= \frac{\cos^2 \theta}{r_x^2} + \frac{\sin^2 \theta}{r_y^2} & c_{yy} &= \frac{\sin^2 \theta}{r_x^2} + \frac{\cos^2 \theta}{r_y^2} \\ c_{xy} &= \frac{\sin \theta \cdot \cos \theta}{r_x^2 \cdot r_y^2}\end{aligned}$$

where r_x and r_y are the extent of the nebula along the major and minor axis, resp., θ is its orientation, and a the intensity falloff.

The specification for this modified model can easily be derived from the one for the Gaussian model shown in Figure 3, essentially by replacing the mean value in the distribution clause (cf. line 17) with the new version of F , and adding declarations for the new model variables. The auxiliary constants and functions are represented as deterministic nodes in the Bayesian network, which are expanded like C-style macro definitions during the program definition. The program for this model is then derived using the same steps as before; the only difference is that the symbolic expressions become much more complicated.

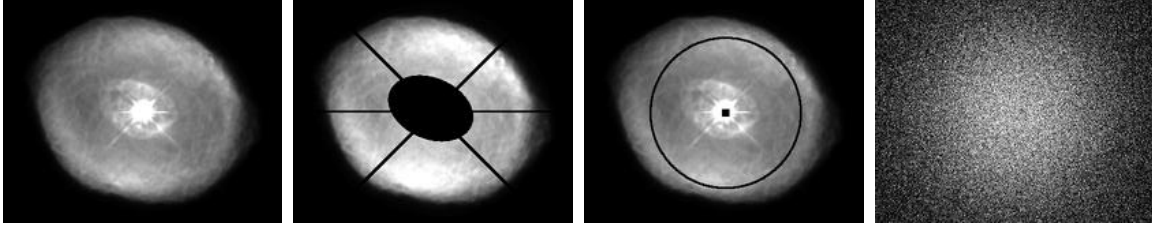


Figure 5. Gaussian analysis of IC418 image: (1) original image (2) manually masked image (3) original image with results superimposed (4) sample data generated from estimated parameter values.

The derivation and resulting program can be simplified and sped up, if the nebula image is assumed to be axis-aligned. This can be modeled by changing the random variable θ into a constant with known value, i.e.,

```
const double theta := 0 as 'orientation'.
```

AUTOBAYES can then propagate this constant value already on the specification level and derive code from the simplified model.

Dual Sigmoidal Model. In a final refinement step, Knuth and Hajian try to estimate the thickness of the shell as well. Since projecting the three-dimensional ellipsoidal shell of gas onto a two-dimensional image produces an ellipsoidal blob surrounded by a ring of higher intensity, the image can be modeled as the difference of two sigmoidal functions with the same center and orientation but different extents, intensities, and falloffs. Re-using the auxiliary definitions from the simple sigmoidal model, this refinement can also be specified easily for AUTOBAYES.

3.3 Model Hierarchy

Knuth and Hajian use the model hierarchy not only to guide the model development but also to guide the parameter estimation itself, i.e., they use the results from the coarser models as starting values for the numerical optimizations in the refined models.

In AUTOBAYES, this effect can be achieved in different ways. The most direct method is to declare the respective parameters in the refined model as `inout` and then to pass the values (manually) from one generated program to the other. A more elegant method is to integrate the models into a single specification which contains a list of task clauses. Then the same program pipeline can be generated from this combined specification; however, this capability is still under development.

4 Image Segmentation Models

In their original analysis, Knuth and Hajian manually masked the central star and the diffraction spikes (cf. Figure 5(2)). This prevents their analysis from misinterpreting

the comparatively bright star as the center of the nebula. In this section we show how AUTOBAYES can be used to derive analysis code which can replace this manual masking step. The basic idea of all models discussed in this section is to segment the image into different conceptual classes (e.g., central star, nebula, and background), and then to use these image-specific classes to replace the generic mask.

4.1 Segmentation via Clustering

It is well-known in data analysis that many images can already be segmented by clustering the pixels just based on their similarities, but without taking into account any spatial information. The simplest model for such a cluster-based segmentation assumes that the image is produced by a mixture of Gaussians in such a way that each pixel is sampled from an independent Gaussian random variable corresponding to the pixel's cluster.

Model Specification. Figure 6 shows the AUTOBAYES-specification for the cluster-based segmentation. At its core is the unobserved (“hidden”) random variable c (cf. lines 12–14) which represents the unknown cluster each pixel belongs to and which determines the cluster's mean and variance (cf. line 16); its values are independently drawn from a discrete distribution with relative frequencies φ , i.e., $pr(c_{ij} = k) = \varphi_k$. The constraint on the probability vector φ (cf. line 11) is required to make this model well-formed. Since we need the cluster-assignments for the segmentation, c is declared as `output`.

Program Derivation. This model is solved by an application of the EM-schema which is triggered by the hidden variable structure of the corresponding Bayesian network. The synthesized EM-algorithm alternates between estimating the hidden variable c (M-step) and the model parameters μ , σ , and φ (E-step) until it reaches convergence. The EM-schema can thus modify the model by marking the hidden variable c as known. AUTOBAYES then recursively solves this modified model, using a number of decomposition schemas enabled by the modification; all emerging subproblems can eventually be solved symbolically (for the detailed derivation cf. [10, 11]).

```

1 model segment as 'Image segmentation via Clustering'.
... (see Figure 3) ...

% Class parameters and relative frequencies
6 const nat n_classes as 'number of classes'.
7 where 0 < n_classes.
8 double mu(1..n_classes), sigma(1..n_classes).
9 where 0 < sigma(_).
10 double phi(1..n_classes).
11 where sum(I:=1..n_classes, phi(I))=1.

% Classes and Distribution
12 output nat c(1..nx, 1..ny) as 'class'.
13 where 1 <= c(_,_) && c(_,_) <= n_classes.
14 c(_,_) ~ discrete(phi).

% Data and Distribution
15 data double pixel(1..nx, 1..ny) as 'image'.
16 pixel(I,J) ~ gauss(mu(c(I,J)), sigma(c(I,J))).

% Task
17 max pr(pixel|{phi,mu,sigma}) for {phi,mu,sigma}.

```

Figure 6. AUTOBAYES-specification for image segmentation model.

Program Results. Figure 7 shows the result of segmenting the IC418 image according to the identified different classes. In particular, a segmentation into three classes already isolates the central star, the nebula, and the background from each other. It can thus be used as a precise, image-specific mask (cf. Figures 5(2) and 7(3)). More classes reveal more details, e.g., identify the nebula’s shell and separate the halo from the background, but too many classes bear the risk of overfitting the image.

4.2 Data Fusion

The segmentation model above can easily be generalized to multivariate data, essentially by changing the data declaration and the corresponding distribution clause to

```

15 data double pixel(1..nc,1..nx, 1..ny) as 'data cube'.
16 pixel(C,I,J) ~ gauss(mu(C,c(I,J)), sigma(C,c(I,J))).

```

and adapting resp. adding some declarations. This allows us to “fuse” multiple images of IC418 which are taken in different wavelengths, resulting in the more detailed and stable segmentations shown in Figure 8. This model is solved by the EM-algorithm in the same way, only that the parameter vectors are replaced by matrices.

4.3 A Refined Segmentation Model

The segmentation models discussed can be refined to take spatial information into account, e.g., by adding geometric background knowledge such as the elliptic shape of the nebula. Such a refinement can be accomplished by a simple hierarchy of models. First, clustering is performed, using the model in Figure 6. Then, all data points belonging to the “shell” class of the nebula are extracted from the data. These data are then analyzed with a model that fits the

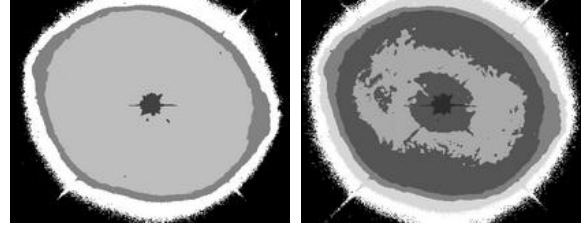


Figure 8. IC418 data cube: (1) five-class segmentation (2) seven-class segmentation.

data to an elliptical ring. This model is very similar to our Gaussian model but uses a slightly different function (d is the “thickness” of the elliptical ring):

$$F(x, y) = i_0 \cdot e^{-\frac{(\sqrt{r(x,y)}-1)^2(r_x^2+r_y^2)}{4d^2}} \quad (3)$$

AUTOBAYES also allows to combine both models into a single specification. We only need to replace the mean $\mu(c(I, J))$ in line 16 of Figure 6 by our function F where all parameters (i_0, r_x, r_y, d) are now vectors over the classes, i.e., they are indexed by $c(I, J)$. AUTOBAYES synthesizes a program with a numerical optimization routine nested within an EM-algorithm. Figure 9 shows sampling data generated with the parameters estimated by that model. Class 1 (white) corresponds to the hull, class 2 (gray) to the core, and class 3 (not shown) to the background.

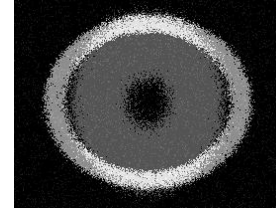


Figure 9. Sample data from refined segmentation model.

5 Evaluation

Table 1 summarizes the results; for each model (referred by the section where it is discussed), it lists the size of the specification and the generated program (*including* generated comments), and the synthesis time (measured on an unloaded 2GHz/4GB LinuxPC). `-nosolve` and `-nolib` are AUTOBAYES command line options which suppress the application of the schemas using partial symbolic solutions and library components, resp., described in Section 3.1.

The table shows that AUTOBAYES’s specification language allows a compact problem representation; none of the

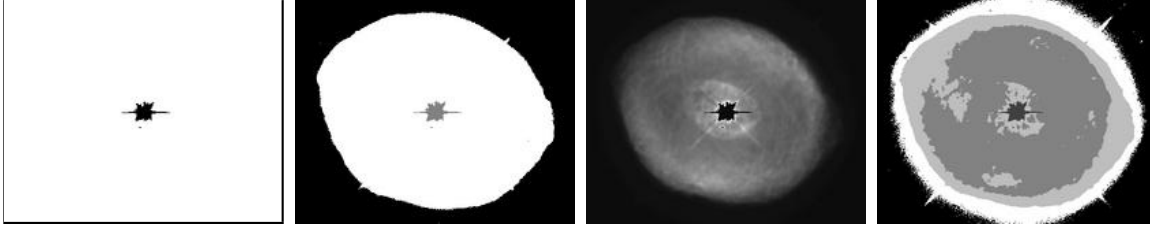


Figure 7. Segmentations of IC418 image: (1) two-class segmentation (2) three-class segmentation (3) ditto, white class used as mask load to original image (4) five-class segmentation.

Model	Spec	Code	T_{synth}
gauss (3.1)	18	1045	36.4s
-nosolve		703	2.4s
-nolib		764	4.9s
-nolib -nosolve		494	1.1s
sigmoid (3.2)	28	-	-
-nosolve		12650	39m42.9s
-nolib -nosolve		872	3.2s
sigmoid-0 (3.2)	27	581	1.3s
sigmoid-2 (3.2)	35	1202	6.7s
segment (4.1)	17	518	1.2s
fusion (4.2)	18	602	1.4s
segment-2 (4.3)	30	1765	40.6s

Table 1. Summary of Results

models requires more than 35 lines. The major difficulty in writing the specifications was to understand and then to express the core idea of the scientists’ models, which was not completely evident from the original analysis. After that, each specification took only a few minutes to write and in general one or two iterations to debug and complete (e.g., adding constraints).

The table also shows the overall feasibility of the approach. AUTOBAYES was able to derive code for each of the models; scale-up factors are generally around 1:30. Synthesis times are generally only a few seconds and comparable to compilation times of the derived code. However, for the gauss and sigmoid models, AUTOBAYES spends almost all time simplifying the partial differentials and then optimizing the auxiliary functions evaluating them; in the sigmoid-case, this even exhausts the available memory. With the command line options, AUTOBAYES can be forced to stay away from these expensive calculations and code is derived much faster, using the Nelder-Mead simplex method which requires no differentials.⁵

Search space explosion is a common problem in program synthesis. In our case, it is mitigated by the deterministic nature of the symbolic-algebraic computations, the higher level of abstraction inherent to the schemas, and the

inherent structure of the schema library. Still, search spaces are large. For the gauss-model, AUTOBAYES derives 224 programs, many of them equivalent, in 35 minutes total synthesis time. Parts of the search space can be pruned away manually by command line options like -nosolve, but more implicit control is required, especially when the schema library grows further.

The scientists’ original Matlab code uses a gradient descent method; it computes the differentials in a single iteration over all pixels while the synthesized code iterates once for each differential but reuses memoized subexpressions. This decomposition requires domain knowledge not yet formalized for AUTOBAYES. For the gauss-model, the (interpreted) Matlab-code is approx. 70 lines, mainly for the computation of the gradient. It requires on average 174 seconds to converge, while the synthesized C++-code only requires 11 seconds.

6 Related Work

Scientific computing is a (relatively) popular application domain for program synthesis; due to its complexity, most related work also follows a schema-based approach. However, most work focuses on wrapping solvers for partial differential equations (PDEs). SciNapse [1] is a general “problem-solving environment” for PDEs; it supports code generation for a variety of features in the PDE-domain, as for example coordinate transformation or grid generation. SciFinance [2] is a domain-specific system for option pricing built on top of SciNapse. Ellman *et al.* [9] describe a system to generate simulation programs from PDEs. All of these systems use Mathematica as underlying symbolic engine. Ctdel [21] is PDE-based synthesis system for the weather forecasting domain. It is also implemented in SWI-Prolog and contains its own symbolic engine.

Other domains than PDEs have been tackled less often. Amphion [19] is a purely deductive synthesis system which has been used to synthesize astronomical software using library components. Amphion/NAV [25] applies the same technology to the state estimation domain. The scale-up difficulties encountered there led to a switch to a schema-based

⁵The table entries for sigmoid-0, sigmoid-2, and segment-2 thus refer to the -nolib -nosolve variants.

approach and the development of the AUTOFILTER-system [24] as a domain-specific extension of AUTOBAYES. It provides its own specification language and schemas but reuses the core system. Planware [4] deductively synthesizes high-performance schedulers. It uses concepts from higher-order logic and category theory to structure the domain theory and thus to reduce the required proof effort.

Schema-based program synthesis is also related to generative programming [6], since schemas (more precisely, the code fragments) correspond to *templates*. The major differences are that (i) the almost completely syntax-directed template instantiation is less powerful and less secure than schema instantiation and (ii) the users still have to write the core algorithm.

Code libraries are common in scientific computing and data analysis, but they lack the level of automation achievable by program synthesis. For example, the Bayes Net Toolbox [15] is a Matlab library which allows program development on the model level, but it does not derive algorithms or generate code. BUGS [20] is a statistical model interpreter based on Gibbs-sampling, a universal—but less efficient—Bayesian inference technique; it could be integrated into AUTOBAYES as an additional schema.

7 Conclusions

We presented AUTOBAYES, a fully automatic program synthesis system for the data analysis domain and demonstrated how it can be used to support a realistic scientific task, the analysis of planetary nebulae images taken by the HST. We specified the hierarchy of models presented in [12]. AUTOBAYES was able to automatically generate code for these models; in tests, the synthesized code gave the same results as the scientists' Matlab code. We also used AUTOBAYES to refine the analysis, combining cluster-based image segmentation with geometric constraints.

Key elements to achieve these results are the schema-based synthesis approach, an efficient problem representation via Bayesian networks, and a strong symbolic-algebraic subsystem. This combination is unique to AUTOBAYES and allows us to solve realistic data analysis problems.

However, AUTOBAYES must still be improved before it can be delivered to the working data analyst. The domain coverage must be extended and schemas must be added to enable solutions for new classes of models (e.g., covariate data). Synthesis from large models requires support for hierarchical specifications and more powerful optimizations to generate faster code. Finally, numeric optimization schemas must be extended with numeric constraint handling to produce more versatile and robust code.

Yet, AUTOBAYES already demonstrates that program synthesis technology has reached a state that enables scientists to think and to program in models rather than in code.

Acknowledgments. Kevin Knuth and Arsen Hajian provided the nebula data and their analysis programs; Kevin also commented on an earlier version of this paper. Wray Buntine contributed substantially to the early development of AUTOBAYES.

References

- [1] R.L. Akers *et al.* SciNapse: A problem-solving environment for partial differential equations. *IEEE Comp.Sci.Eng.*, 4(3):33–42, 1997.
- [2] R.L. Akers *et al.* SciFinance: A program synthesis tool for financial modeling. *AI Magazine*, 22(2):27–41, 2002.
- [3] A. Ayari and D. Basin. A higher-order interpretation of deductive tableau. *JSC*, 31(5):487–520, 2001.
- [4] L. Blaine *et al.* Planware – domain-specific synthesis of high-performance schedulers. In *ASE-13*, pp. 270–280. IEEE, 1998.
- [5] W.L. Buntine. Operations for learning with graphical models. *JAIR*, 2:159–225, 1994.
- [6] K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2002.
- [7] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society Series B*, 39:1–38, 1977.
- [8] E. Denney and B. Fischer. Correctness of source-level safety policies. In *FM-2003*. To appear.
- [9] T. Ellman, R. Deak, and J. Fotinatos. Knowledge-based synthesis of numerical programs for simulation of rigid-body systems in physics-based animation. In *ASE-17*, pp. 93–104. IEEE, 2002.
- [10] B. Fischer and J. Schumann. AutoBayes: A system for generating data analysis programs from statistical models. *JFP*, 2003. To appear.
- [11] A.G. Gray, B. Fischer, J. Schumann, and W. Buntine. Automatic derivation of statistical algorithms: The EM family and beyond. In *NIPS-15*. MIT Press, 2002.
- [12] K.H. Knuth and A.R. Hajian. Hierarchies of models: Toward understanding of planetary nebulae. In *Bayesian Inference and Maximum Entropy Methods in Science and Engineering 22*, pp. 92–103, 2002.
- [13] Z. Manna and R. Waldinger. *The Deductive Foundations of Computer Programming*. Addison-Wesley, 1993.
- [14] G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997.
- [15] K. Murphy. The Bayes Net Toolbox for Matlab. *Computing Science and Statistics*, 33, 2001.
- [16] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer. CIL: Intermediate language and tools for analysis and transformation of C programs. In *11th Intl. Conf. Compiler Construction, LNCS 2304*, pp. 213–228. Springer, 2002.
- [17] K. Olmos and E. Visser. Strategies for source-to-source constant propagation. In *Intl. Workshop Strategies in Rewriting and Programming, ENTCS 70.6*. Elsevier, 2002.
- [18] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [19] M. Stickel *et al.* Deductive composition of astronomical software from subroutine libraries. In *CADE-12, LNAI 804*, pp. 341–355. Springer, 1994.
- [20] A. Thomas, D.J. Spiegelhalter, and W.R. Gilks. BUGS: A program to perform Bayesian inference using Gibbs sampling. In *Bayesian Statistics 4*, pp. 837–842. Oxford Univ. Press, 1992.
- [21] R.A. van Engelen, L. Wolters, and G. Cats. Ctdel: A generator of efficient code for PDE-based scientific applications. In *9th Intl. Conf. Supercomputing*, pp. 86–93. ACM, 1996.
- [22] E. Visser. Scoped dynamic rewrite rules. In *Intl. Workshop Rule-Based Programming, ENTCS 59.4*. Elsevier, 2001.
- [23] M. Whalen, J. Schumann, and B. Fischer. Synthesizing certified code. In *FME 2002, LNCS 2391*, pp. 431–450. Springer, 2002.
- [24] J. Whittle and J. Schumann. Automating the implementation of Kalman-filter algorithms. In review.
- [25] J. Whittle *et al.* Amphion/NAV: Deductive synthesis of state estimation software. In *ASE-16*, pp. 395–399. IEEE, 2001.